ETA: Experience with an Intel Xeon Processor as a Packet Processing Engine

AS PART OF AN EFFORT TO ACCELERATE SERVER TCP/IP NETWORKING, INTEL R&D HAS DEVELOPED A SOFTWARE PROTOTYPE THAT USES ONE OF THE INTEL XEON PROCESSORS IN A MULTIPROCESSOR SERVER AS A PACKET PROCESSING ENGINE (PPE). THIS PROTOTYPE SERVES AS A VEHICLE FOR EMPIRICAL MEASUREMENT AND ANALYSIS OF A HIGHLY PROGRAMMABLE PPE THAT IS CLOSELY TIED TO THE SERVER'S CORE CPU AND MEMORY COMPLEX.

Greg Regnier
Dave Minturn
Gary McAlpine
Vikram A. Saletore
Annie Foong
Intel

Server-based networks have well-documented performance limitations.[1,2] These limitations outline a major goal of Intel's Embedded Transport Acceleration (ETA) project, the ability to deliver high-performance server communication and I/O over standard Ethernet and Transmission Control Protocol/Internet Protocol (TCP/IP) networks. By developing this capability, Intel hopes to take advantage of the large knowledge base and ubiquity of these standard technologies. With the advent of 10 gigabit Ethernet, these standards promise to provide the bandwidth required of the most demanding server applications. In addition, a substantial increase in the performance of server networks could let standard high-volume servers perform many storage and communication-centric applications commonly served by specialized appliances.

We use the term *packet processing engine* (PPE) as a generic term for the computing and memory resources necessary for communication-centric processing. Such PPEs have certain desirable attributes; the ETA project focuses on developing PPEs with such attributes, which include scalability, extensibility, and programmability.

A PPE must scale in terms of communication throughput as well as in its ability to simultaneously support many sessions. Extensibility makes it possible to add value to the solution over time in terms of new features, protocols, and applications. Programmability helps in adapting a PPE to changing standards and in modifying its behavior in subtle but important ways. These desired attributes tend to lead to a solution that does not artificially constrain the amount of processing and memory resources.

General-purpose processors, such as the Intel Xeon in our prototype, are extensible and programmable by definition. The results from our prototype show that the performance and scalability of the general-purpose processor can be improved as well. For future CMP processors, which will integrate multiple processing cores into a single chip and have increasingly abundant processing resources,

Published by the IEEE Computer Society

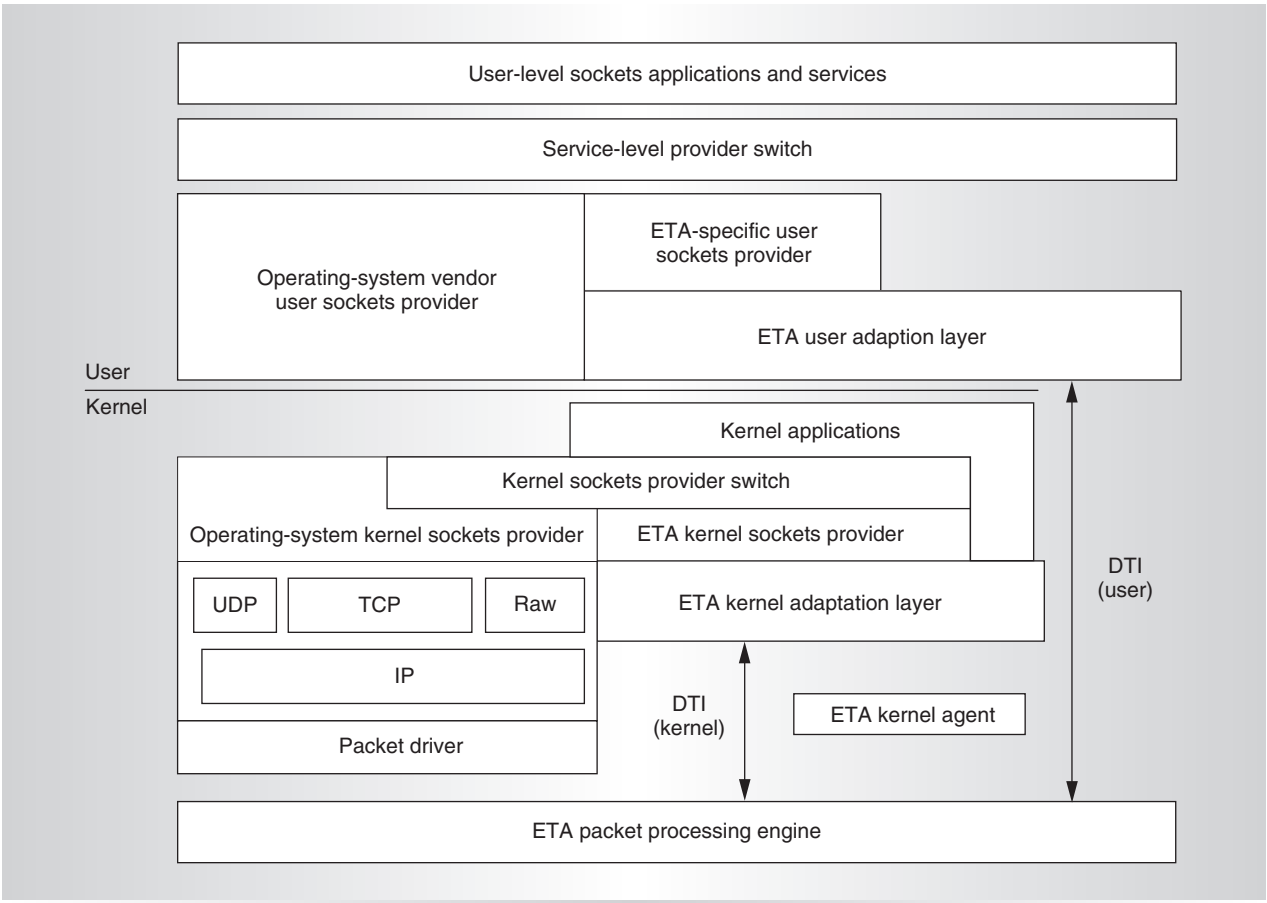

Figure 1. ETA host software stack.

using a general-purpose processor as a PPE becomes increasingly tenable for network-intensive applications. We are developing ETA with these ideas in mind.

ETA software architecture

At a high level, the ETA architecture partitions the server software between host and PPE processing resources. The host is where the general-purpose operating system and applications reside. The PPE is where the communication-centric tasks, including network protocol processing, are performed. We implement the interface between the host and the PPE as a set of asynchronous queues in a cache-coherent, shared host memory. These queuing structures support control, synchronization, and the receipt and transmission of data.

ETA host software

The ETA host's software stack allows for multiple paths between host applications and the PPE. The stack also supports accelerated networking paths for applications at both the kernel and user levels, and there is also a nonaccelerated path through the operating system's native TCP/IP and driver stack. At both the kernel and user levels, a thin layer of software acts as an adaptation layer, providing an asynchronous programmatic interface to queuing structures that form the interface between the host and the PPE. In addition, ETA works with legacy sockets applications as well as new applications written directly to ETA-specific interfaces. Figure 1 shows a high-level view of the ETA host software stack.

ETA host-engine interface

The interface between the host processors and the PPE is implemented as a set of queuing structures called direct transport interfaces. DTIs are based on the principles of the Virtual Interface[3] and the InfiniBand (http://www.infinibandta.org) architectures,

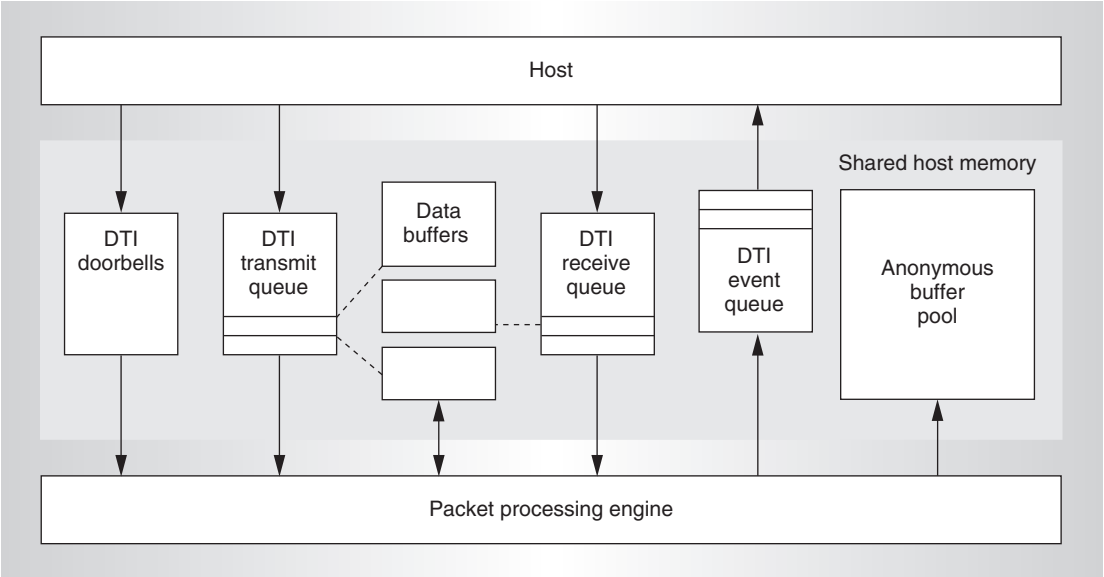

Figure 2. DTI queuing structures.

but differ in their optimization for IP networking semantics. In particular, the DTI structures also support TCP connection commands in addition to data transmit and receive. DTIs also provide anonymous buffer pools to support the buffering semantics of TCP streams. Figure 2 shows the structure of the DTI queuing structures.

Each DTI can include a send queue, a receive queue, event queues, doorbells, and data buffers in an associated buffer pool. Individual DTIs include all of these elements. However, we can create groups of DTIs for any given server application such that they share a common event queue and set of doorbells. Each child DTI can include only send and receive queues, and associated data buffer pools. Parent DTIs "listen" on new TCP connections. When the ETA host accepts and establishes a new connection, the parent DTI associates a child DTI with the new connection; a child DTI's main function is to service a TCP session. The following describes the DTI elements:

- The *ETA host* uses send and receive queues to post send and receive descriptors to the ETA PPE. The data transfers directly to or from application buffers or the DTI anonymous buffer pool.
- The *event queue* enables the ETA PPE to post event notices to the host application. Each DTI can have a private event queue or have its event notices directed to an event queue shared by multiple DTIs.
- The DTI *data buffers* enable the ETA PPE to buffer data for the DTI in one of three situations: when the source or target application buffers are not preconditioned; when the PPE receives TCP segments and there are no receive descriptors posted on the receive queue; or when the PPE receives TCP segments out of order.
- The ETA host processors use the *doorbell queue* to write notices or signals directly to the ETA PPE and indicate a context associated with each notice or signal.

ETA PPE software

The ETA architecture is largely independent of the PPE implementation, which can be either a fixed device, a specialized programmable engine, or as in our prototype's case, a general-purpose CPU. An ETA-aware PPE must support several specific functions.

First, it must support DTI queuing structures by working with notices of new work posted on the send and receive queues via the doorbell queue. In addition, the PPE must support the event queue and be able to interrupt the host processors when an application is blocked, waiting for a transaction to complete.

The PPE must also execute the actual packet processing functions on behalf of the host,

and (of course) support an interface to the network itself. In the case of our prototype, the packet processing functions are mainly the termination of TCP/IP connections on behalf of server applications.

ETA prototype

Our ETA prototype uses a dual-processor server that contains two Intel Xeon CPUs. The prototype uses one CPU as the host processor and one as the PPE. The PPE's main function is to establish and terminate TCP/IP sessions on behalf of applications running on the host CPU.

This configuration has several advantages. First and foremost, there was no special hardware to develop. Secondly, we could use standard tools to develop the software for both the host and the PPE. We also use standard gigabit Ethernet network cards with a modified version of the Ethernet driver. Finally, we can use shared, coherent memory to implement the interfaces between the host CPU and the PPE.

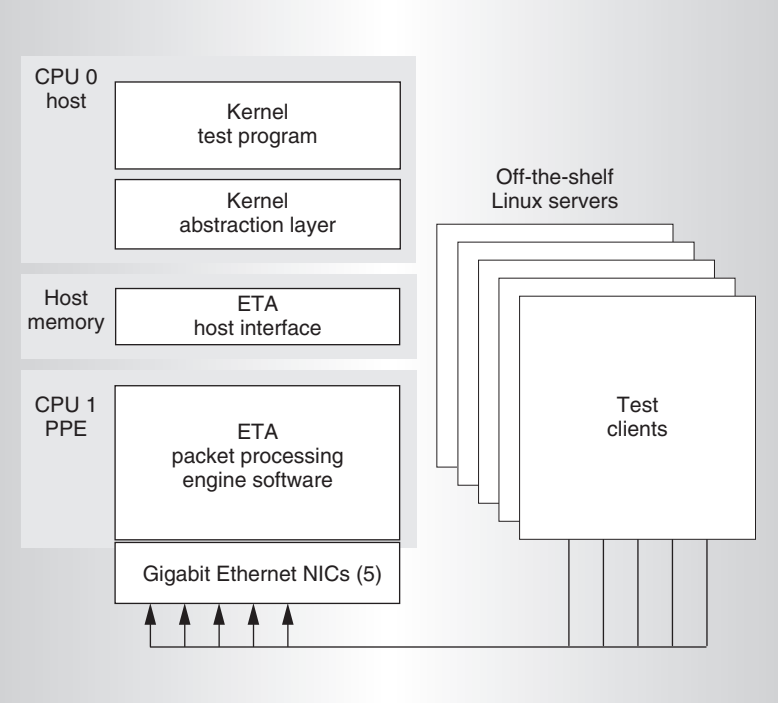

Figure 3. ETA prototype test environment.

Prototype software environment

We developed the prototype using a standard Linux kernel, version 2.4. The PPE software is basically a loadable Linux module with a stripped-down kernel TCP/IP stack. We added code to support the DTI interfaces and also added a modified gigabit Ethernet driver. The PPE software module has affinity for one processor (CPU1) on the dual-processor platform and never yields control of the processor. This enables using CPU1 as a dedicated PPE.

Synchronization is a unique aspect of the prototype's PPE software. We modified the interface to the network interface card (NIC), disabling the use of interrupts for data transfer operations. Instead, the PPE can poll on NIC descriptors in shared host memory to detect completed packet transactions. Communication between the host CPU and the PPE via the DTI structures also occur through the DTI doorbells in shared host memory. Thus, the PPE can poll the doorbell addresses and NIC descriptors for synchronization without causing memory bus traffic until it becomes necessary to modify the cache lines of the associated shared memory. This type of communication lets the PPE run without interrupts and avoid the associated overheads.

Prototype hardware platform

The prototype can run on virtually any multiprocessor platform that runs the Linux kernel. Our dual-processor platform uses two 2.4-GHz Xeons on a 400 MHz front-side bus. The NICs are standard Intel Pro1000 gigabit Ethernet controllers.

Prototype test environment

Our test environment consists of the server under test (the ETA prototype server) and five client computers connected directly by gigabit Ethernet links. The client computers are standard off-the-shelf servers running Linux and the Test TCP (TTCP) throughput microbenchmark.

The tests running on the ETA prototype are kernel-level applications that interface directly to the ETA kernel abstraction layer. Figure 3 shows the basic test environment.

Measurement and analysis results

We performed basic throughput tests on the ETA prototype, transmitting and receiving data in several data transfer sizes. For comparison, we also ran the TTCP throughput microbenchmark on a standard Linux dual-processor server.

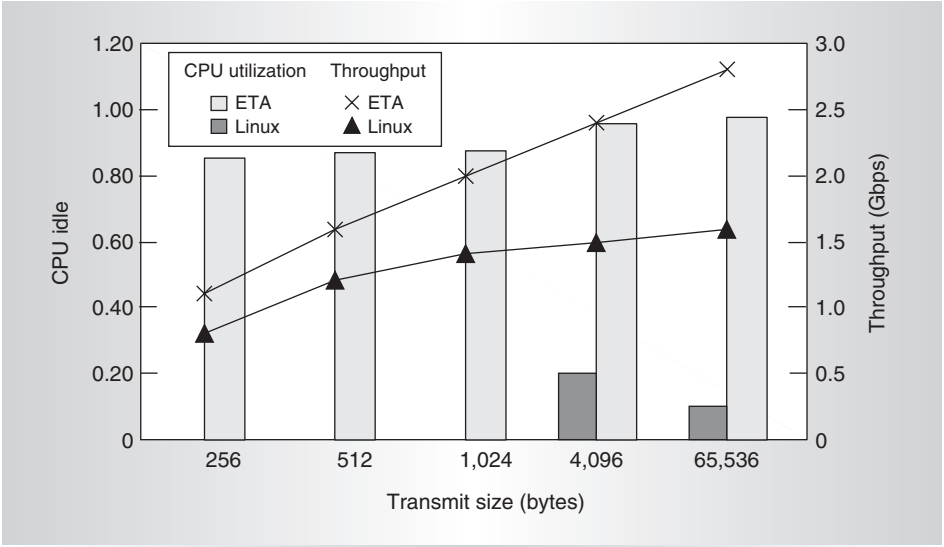

Figure 4. Transmit performance.

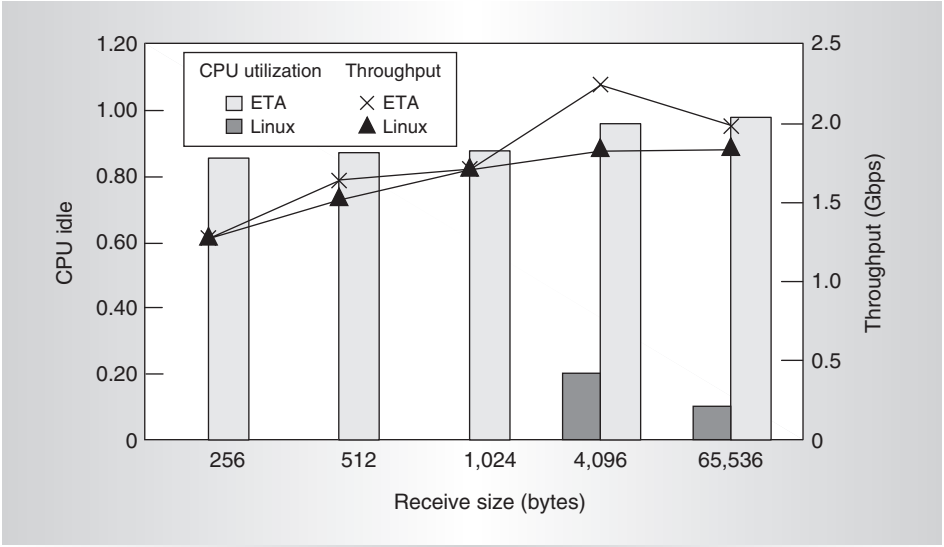

Figure 5. Receive performance.

Figure 4 shows transmit performance along with the portion of the CPU that is idle and thus available for application use. For transfers of 1,024 bytes and less, both CPUs of the standard Linux server were 100 percent utilized in executing the networking stack, thus leaving zero CPUs idle. For larger messages, 20 percent or less of one of the CPUs remained idle.

For the ETA server, the host CPU was less than 20 percent utilized across all transfer sizes, leaving more than 80 percent of one CPU idle and available. In addition, the ETA transmit throughput considerably exceeded that of the standard Linux server for all data transfer sizes.

Figure 5 shows throughput and available CPUs for the receive path. For all cases, both CPUs of the standard Linux server were 100 percent utilized running the networking stack, thus leaving zero CPUs for other applications (that is why the dark gray bars don't show on the graph). The ETA prototype's host CPU was less than 20 percent utilized for all receive transaction sizes, leaving 80 percent of a CPU idle and available. ETA receive throughput exceeds that of the Linux server by a relatively small margin. This smaller improvement in transmission is partly due to memory-to-memory copy performance. Our ETA implementation uses a one-copy receive path because we use off-the-shelf NICs that place packets directly into a preallocated packet buffer. The PPE must then copy these packet buffers to destination buffers.

Figure 6 compares the transmit performance of the standard Linux server (a dual processor, symmetric multiprocessor, denoted 2P SMP), the ETA prototype in single-threaded mode (ETA ST), and the ETA prototype with Intel's hyperthreading technology[4] enabled on the packet processing engine (ETA HT). With hyperthreading enabled, the PPE runs on two hardware threads and provides a degree of parallelism, thus hiding the memory latency that we have found to be a performance limiter of TCP/IP processing on servers. The results show an approximately 50 percent increase in transmit performance on the ETA prototype with hyperthreading, achieving more than 4 Gbps throughput.

Figure 7 shows receive throughput for the

same three implementations plus one more. We added a test path where we enabled hyperthreading but did not execute the data copy on the PPE (ETA HT NoCopy). For receive, we see that enabling hyperthreading improved the ETA performance by about 20 percent. As we noted before, this relatively small improvement comes partly from the copy performance on our prototype implementation. When we omit the copy by the PPE into the test application buffer, we see a significant performance increase, similar to the performance of the transmit case (nearly 4 Gbps).

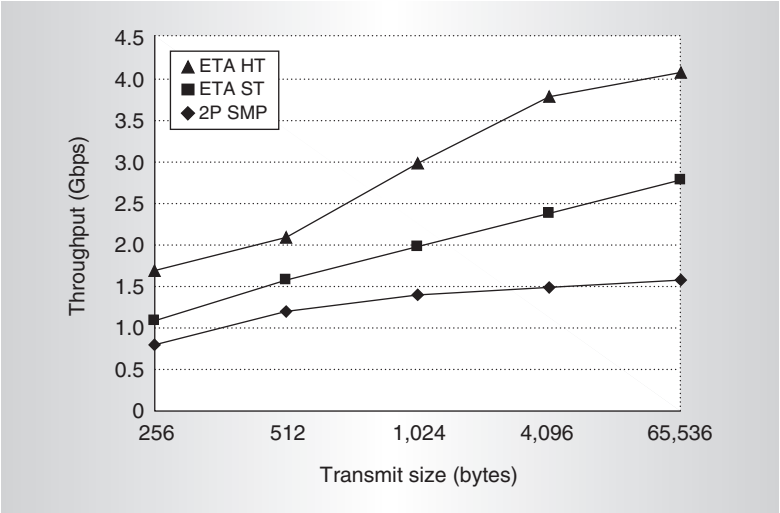

Figure 6. Transmit performance with hyperthreading.

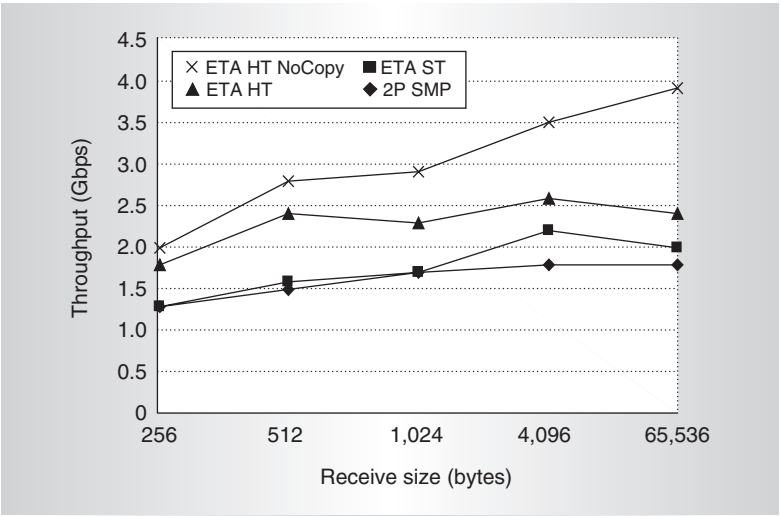

Figure 7. Receive performance with hyperthreading.

Related work

Recent commercial efforts to increase server network performance have centered on specialized TCP/IP Offload Engine devices.[5,6] TOE devices generally offload varying amounts of the TCP/IP protocol stack onto a device that attaches to the server's I/O subsystem. TOE devices generally use separate, specialized processing and memory resources. The ETA prototype described here differs from these devices in that it uses the processing and memory resources of the server itself, making the PPE a first-class citizen of the core CPU and memory complex. We also partition the software so that it is much more efficient than in standard symmetric multiprocessing systems.

Other related research efforts include the Queue Pair IP work at Berkeley.[7] This project showed the effectiveness of interfacing IP protocols implemented on an intelligent network adapter using the InfiniBand architecture's Queue Pair model. The TCP Servers project at Rutgers University developed a framework that could partition network processing onto a dedicated node, processor, or an intelligent adapter.[8] This scheme used lightweight communication mechanisms for the host application interfaces.

Our results show that software partitioning can significantly increase the overall communication performance of a standard multiprocessor server. Specifically, partitioning the packet processing onto a dedicated set of compute resources allows for optimizations that are otherwise impossible when time sharing the same compute resources with the operating system and applications. For example, our prototype PPE does not incur the overhead of interrupts and system calls because it can poll shared memory for new work. It can poll without placing load on the memory subsystem or front-side bus because our architecture relies on the cache coherence protocols to allow the PPE to poll internally in cache. The PPE largely avoids cache interference because it does not share caches with the operating system and applications except through the ETA host interface. Other optimizations are possible, such as strategic prefetching of control and packet header information.

We have seen that threading—Intel's hyperthreading technology in particular for our prototype—is an important factor in increasing

performance. Multithreading allows parallelism that is useful for hiding memory latency. Networking workloads tend to have poor locality and thus poor cache behavior. For example, when a device receives a new packet, it never goes into the cache. So the PPE's reference of that packet incurs a significant memory latency. Given the growing disparity between processor speed and memory latency, multithreading will become more important over time.

The ETA model of software partitioning logically extends to CMPs, which integrate multiple processing cores onto a single die and incorporate increasingly abundant server processing resources. Specialized packet processors with support for specific networking functions have the potential for providing even greater absolute performance.[9] The type of processing and memory resources used for packet processing involve a set of tradeoffs in terms of absolute performance, flexibility, extensibility, and cost.

Our additional ETA development currently focuses on the capabilities of our PPE to accelerate TCP connection establishment. This capability is important for Web servers that deal with many short-lived connections. We are also developing and analyzing an IP storage stack (iSCSI) over ETA and are investigating techniques to minimize data copies through the use of the Direct Data Placement (DDP) and Remote Direct Memory Access (RDMA) protocols currently under definition. Other work focuses on enabling legacy user-level sockets applications on ETA; once complete, this work will let us run and analyze many network applications. MICRO

References

1. J. Kay and J. Pasquale, "The Importance of Non-Data Touching Processing Overheads in TCP/IP," *Conf. Proc. Communications, Architectures, Protocols and Applications*, ACM Press, 1993, pp. 259-268.
2. A. Foong et al., "TCP Performance Re-Visited," *Proc. 2003 IEEE Int'l Symp. Performance Analysis of Systems and Software* (IPASS 03), IEEE Press, 2003, pp. 70-79.
3. D. Cameron and G. Regnier, *The Virtual Interface Architecture*, Intel Press, 2002.
4. W. Magro, P. Peterson, and S. Shah, "Hyper-Threading Technology: Impact on Compute-Intensive Workloads," *Intel Technology J.*, Feb. 2002, http://www.intel.com/technology/itj.
5. L. Gwennap, "Count on TCP Offload Engines," *EETimes*, 2001; http://www.eetimes.com/semi/c/ip/OEG20010917S0051.
6. P. Sarkar, S. Uttamchandani, and K. Voruganti, "Storage over IP: When Does Hardware Support Help?" *Proc. 2nd Usenix Conf. File and Storage Technologies*, Usenix Assoc., 2003.
7. P. Buonadonna and D. Culler, "Queue-Pair IP: A Hybrid Architecture for System Area Networks," *Proc. Int'l Symp. Computer Architecture* (ISCA 02), IEEE CS Press, 2002, pp. 247-256.
8. M. Rangarajan et al., *TCP Servers: Offloading TCP Processing in Internet Servers*, tech. report DCS-TR-481, Dept. of Computer Science, Rutgers Univ., Mar. 2002.
9. Y. Hoskote et al., "A 10GHz TCP Offload Accelerator for 10Gbps Ethernet in 90nm Dual-VT CMOS," *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 03), IEEE Press, 2003, pp. 258-268.

Greg Regnier is a principal engineer in Intel's Corporate Technology Group. His industry experience includes massively parallel computing systems, such as the FPS T-Series, the Touchstone Delta Supercomputer, and the Intel Paragon Supercomputer; he was a principal developer of the Virtual Interface Architecture. His current research interest is improving the scalability and performance of data center networks using standard networking media and protocols. Regnier has a BS in computer science from St. Cloud State University in Minnesota. He is a member of IEEE Computer Society.

Dave Minturn is a senior architect and software engineer in Intel's Corporate Technology Group. His industry experience includes work on parallel supercomputers, I/O subsystems, networking protocols, network processors, and file systems; current research interests include investigating threading models and other optimizations to efficiently scale TCP/IP packet processing on IA-32 processors. Minturn has a BS in computer science from Northern Arizona University.

Gary McAlpine is a system architect in Intel's Communications Technology Lab. His industry experience includes the founding of a company, Aptec Computer Systems, where he developed high-speed multiprocessor systems for use in high-end data collection and real-time applications for signal and image processing. At Intel, he helped develop the Virtual Interface architecture, NGIO (Next-Generation I/O, an InfiniBand precursor), PCI-Express Advanced Switching, and ETA.

Vikram A. Saletore is a staff software engineer with Intel Research and Development, where he works on kernel- and user-level ETA. His research interests include high-speed TCP, network interfaces, operating systems, and distributed computing. Saletore has an MS in electrical engineering from the University of California, Berkeley, and a PhD in electrical engineering from the University of Illinois, Urbana-Champaign. He is a Member of the IEEE Computer Society.

Annie Foong is a senior research engineer at Intel Research and Development. Her research interests include implementation issues for improving network protocol performance. Foong has a PhD in electrical and computer engineering from the University of Wisconsin-Madison.

Direct questions and comments about this article to Greg Regnier, Intel Corp., JF3-232, 2111 NE 25th Ave., Hillsboro, OR 97124; greg.j.regnier@intel.com.